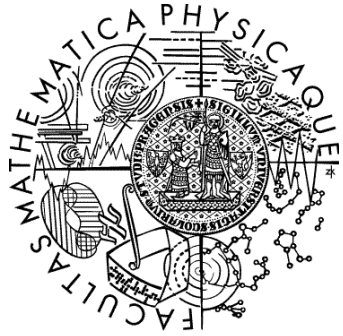
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Stanislav Kozina

Dispatcher pro Workflow Management system

Středisko informatické sítě a laboratoří

Vedoucí bakalářské práce: Dan Lukeš

Studijní program: Informatika, Správa počítačových systémů

2008

Rád bych na tomto místě poděkoval Danu Lukešovi, ing. Tomáši Benedovi a firmě Teledin s.r.o. Bez nich by tato práce nevznikla.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 30.5.2008 Stanislav Kozina

Obsah

Název práce: Dispatcher pro Workflow Management System

Autor: Stanislav Kozina

Katedra: Středisko informatické sítě a laboratoří

Vedoucí bakalářské práce: Dan Lukeš

e-mail vedoucího: dan@mff.cuni.cz

Abstrakt: Předložená práce se zabývá zpracováním a identifikací zpráv standardů UN/EDIFACT, Inhouse a ebXML. Základní cíle práce jsou dva. Prvním cílem je navrhnout formát šablon popisujících strukturu zpráv a umístění identifikačních údajů ve zprávě. Druhým cílem je pak navrhnout a implementovat systém pro zpracování a extrakci informací ze zpráv podle těchto šablon. Šablony pro jednotlivé standardy by měly být navzájem co nejvíce podobné, dobře čitelné i zapisovatelné pro člověka, a musí být schopny popsat libovolnou strukturu zprávy. Informace, které byly ze zprávy po zpracování vyjmuty, jsou následně zapsány do nálepky ke zprávě, která slouží ke snazší identifikaci zprávy při dalším zpracování.

Klíčová slova: EDI, UN/EDIFACT, ebXML, Parser

Title: Dispatcher for Workflow Management system

Author: Stanislav Kozina

Department: Network and Labs Management Center

Supervisor: Dan Lukeš

Supervisor's e-mail address: dan@mff.cuni.cz

Abstract: Present work considers processing and identification of the messages under UN/EDIFACT, Inhouse or ebXML standard. There are two main goals of this work. The first goal is to design a template format describing the structure of the messages and the location of identification information in the message. The second goal is to design and implement system for processing and information extraction from the message according to the template. Templates itself should be very similar for all three standards, should be easily read and written by human and must be able to describe any structure of the message. Information which where picked up from the message are subsequently written in the message sticker, which serves for easier identification in later processing.

Keywords: EDI, UN/EDIFACT, ebXML, Parser

1 Úvod

Posláním celého Workflow Management System je převzetí výměny od odesilatele, její zpracování a předání příjemci. **Výměnou** (Interchange) rozumějme určité surové množství dat, kterou si mezi sebou vyměňují jednotlivé subjekty. Celá data jsou organizována podle jednoho ze standardů UN/EDIFACT, ebXML, Inhouse či dalších. **Zprávou** pak rozumějme již identifikovanou část výměny – data, u kterých přesně známe formát, v jakém jsou zapsána, a jsme schopni určit sémantický význam těchto dat. Výměna může sestávat hned z několika zpráv stejného standardu. Zpráva se dále skládá ze **segmentů** – částí, které reprezentují jednotlivá data. Jednotlivé segmenty se ještě mohou skládat, v některých případech, z několika **elementů**.

Převzetím výměny rozumíme příjem jedním z několika používaných komunikačních prostředků. Nazývejme jej **vstupní kanál**. Dnes se jedná nejčastěji o poštu dle protokolu X.400, peer-to-peer protokol OFTP a zabezpečené spojení protokolem FTP Edirex.

Zpracováním výměny rozumíme přesnou identifikaci výměny, například jejího odesilatele, příjemce a typ zprávy, a případné provedení dalších úkolů. Často dochází ke konverzi výměn mezi různými standardy, může dojít také k zašifrování či podepsání výměny elektronickým podpisem. Při všech těchto úkolech nesmí dojít ke změně obsahu zprávy, nekontroluje se její faktický význam.

Předáním výměny příjemci rozumíme opět odeslání výměny jedním z používaných komunikačních prostředků. Tento prostředek nazývejme **výstupní kanál**. Platí pro něj obdobná pravidla jako pro kanál vstupní.

Pro korektní zpracování zprávy je nutné znát různé informace o zprávě. Mezi tyto informace patří standard zprávy, její odesilatel, příjemce, druh obsahu a další. Často by nám přišla vhod také data vyjmutá z mnoha různých míst zprávy. Získání těchto informací nazvěme proces **identifikace** zprávy.

Doposud probíhá identifikace zpráv velice jednoduchým způsobem. Většina kanálů je schopna poskytnout dostatek informací potřebných k identifikaci zprávy. Z některých kanálů jsou zase přijímány pouze zprávy od jednoho odesílatele. V těchto případech je celý proces snadný.

V některých případech se informace nutné k identifikaci zprávy nacházejí na předem dohodnutém místě. Pro různé druhy zpráv pak existují různé nástroje, které tyto informace ze zpráv získávají. Ve výjimečných případech jsou ale informace skryty hluboko ve zprávě, a jejich vyhledání je pak velice obtížným úkolem. Dosavadní systém takové zprávy často předává lidem k ručnímu zpracování.

Z těchto důvodů je nutné vyvinout nový jednotný modul, který by byl schopen získávat ze všech zpracovávaných druhů zpráv dostatek informací potřebných k identifikaci zprávy. Tomuto modulu říkejme modul Dispatcher, protože vyčkává na vstupním kanálu a rozhoduje, jak bude se kterou zprávou naloženo. Jeho úkolem je identifikace výměny, tedy sestavení jakési množiny informací, díky které lze snadno přesně identifikovat zprávu. Této množině informací říkejme **nálepka**. Alternativním názvem by mohla být průvodka, tento pojem má ale již v celém systému lehce jiný význam. Nálepka může být reprezentována souborem, nebo třeba záznamem v databázi.

Jelikož se podoba a formát zpracovávaných zpráv časem mění, je nutné mít předem k dispozici popis struktury všech zpráv a zároveň umístění všech informací ve zprávě,

které jsou relevantní pro korektní identifikaci zprávy. Tomuto popisu struktury říkejme **šablona** zprávy.

Kapitola 2 obsahuje jednoduché představení všech zpracovávaných standardů zpráv. Tato kapitola rozhodně neobsahuje vyčerpávající popis těchto standardů, je ale dobrým přehled o základních prvcích jednotlivých formátů.

Kapitola 3 se zabývá návrhem a popisem formátu šablon zpráv. Návrh šablon schopných popsat všechny zpracovávané zprávy je jedním z úkolů této práce.

Kapitola 4 popisuje postup práce Dispatchera. Obsahuje popis algoritmu běhu celého modulu, stejně jako popis procesu porovnávání zpráv podle dostupných šablon.

Kapitola 5 obsahuje základní uživatelskou dokumentaci. Zde je popsáno jak modul správně nakonfigurovat a spustit. Součástí práce uživatele je ale také tvorba a údržba šablon zpráv, uživatel se proto musí seznámit také s Kapitolou 3.

Kapitola 6 je tvořena programátorskou dokumentací. Zde čtenář nalezne popis hlavních struktur programu, popis konkrétní realizace a podmínky pro překlad a spuštění programu.

Kapitola 7 obsahuje závěr, stručné shrnutí celé práce a možného budoucího vývoje modulu. Slovníček v Kapitole 8 obsahuje nejdůležitější pojmy, které se v práci vyskytují. Obsah přiloženého CD je popsán v závěrečné Kapitole 9.

2 Popis standardů zpracovávaných zpráv

Před popisem vlastního modulu Dispatcher je nutné seznámit se se zpracovávanými standardy zpráv. Nelze bohužel dopředu určit formát všech zpráv, které budou systémem procházet. Většina zpráv je ale jednoho ze tří základních standardů.

2.1 Standard UN/EDIFACT

Zkratka UN/EDIFACT stojí za názvem United Nations / Electronic Document Interchange For Administration, Commerce and Transport. Standard byl adoptován světovým standardizačním institutem jako standard ISO 9735. V českém prostředí pak vyšla tato norma jako ČSN ISO 9735: Elektronická výměna dat pro správu, obchod a dopravu (EDIFACT).

Zpráva standardu EDIFACT je tvořena segmenty, která na sebe přímo navazují. Každý segment končí daným koncovým znakem. První tři znaky segmentu udávají jméno (návěští) segmentu. Uvnitř segmentu se nachází několik elementů, jednoduchých či složených, oddělených dohodnutými separátory. Tabulka 1: Výchozí EDIFACT separátory obsahuje přehled výchozích separátorů a dalších speciálních znaků:

Znak	Význam znaku
:	Separátor prvků složeného elementu
+	Separátor elementů
.	Desetinná tečka
?	Zprošťující znak
*	Separátor opakujících se elementů (od verze 4)
'	Koncový znak segmentu

Tabulka 1: Výchozí EDIFACT separátory

Tyto separátory a speciální znaky je možné přetížit, tedy lze použít místo výchozích znaků znaky jiné. Prvním segmentem ve zprávě může být nepovinný segment UNA, který dále obsahuje právě 6 znaků – těchto 6 uvedených separátorů ve stejném pořadí, v jakém jsou v tabulce. V EDIFACT zprávě verze 3 a nižší se vynechá znak pro separaci opakujících se elementů (hvězdička). Tento znak ve zprávě nemá význam separátoru, v segmentu UNA se na jeho místě uvádí znak mezera.

V samotné zprávě jsou segmenty zapsány volně za sebou. Jejich sémantický význam, respektive význam jejich obsahu, je ale dán dohodou odesílatele a příjemce. Jednotlivé segmenty se často po skupinách opakují. V praxi můžeme rozlišovat několik **typů zpráv**, podle dohodnutého obsahu a pořadí segmentů ve skupinách. Pro každý typ existuje přesný předpis struktury zprávy. **Strukturou** zprávy tedy rozumějme předem dané pořadí segmentů, povinností segmentů a sémantický význam dat obsažených v těchto segmentech. Struktura zprávy je dána **normou** zprávy. Například pro zprávu reprezentující fakturu existuje norma INVOIC.

Některé segmenty ve zprávách mohou být povinné, jiné nepovinné. U každé skupiny je v normě uveden maximální počet opakování skupiny.

Velice často se budeme snažit ze zprávy získat jejího odesilatele a příjemce. Podle normy ISO 9735 je kód odesilatele i příjemce uveden jako druhý a třetí element segmentu záhlaví výměny UNB. Jenže zde je uveden poslední odesilatel – pokud ten pouze předává zprávu jiného vydavatele, může pro nás být mnohem hodnotnější identifikace tohoto skutečného tvůrce zprávy. A jeho identifikace může být skryta kdekoli ve zprávě.

Vlastní formát zpráv standardu EDIFACT je sice dán normou, nelze se ale na ni plně spolehnout. Například mezi koncovým znakem segmentu a prvním znakem následujícího segmentu dle normy nesmí být žádný znak. V praxi se sem ale dává několik bílých znaků, například znak konce řádky příslušné platformy pro lepší čitelnost celé zprávy. Nutností pro zpracování EDIFACT zpráv je proto praktická zkušenost.

Podle normy jsou zprávy standardu EDIFACT přenášeny v 7-bitovém kódu podle normy ISO 646 (ASCII), pokud se účastnící nedohodli na použití 8-bitového kódu podle norem ISO 6937, ISO 8859 či jiných. V praxi tedy můžeme potkat zprávy v různých znakových sadách.

2.2 Standard ebXML

Standard ebXML bohužel není korektně specifikován. Jako jistá náhrada může sloužit soubor norem ISO 15000: Electronic business eXtensible markup language. EbXML zprávy jsou ale starší než tento soubor norem, proto se mohou některé ze zpráv z těchto norem vymykat. Stejně tak úplné doporučení konsorcia W3C pro formát XML dokumentů je novější než ebXML standard. EbXML zprávy proto dokonce nemusí splňovat podmínky takzvaného „well-formed“ XML. Pro práci s ebXML zprávami jsou velice důležité praktické zkušenosti, neboť pouze zkušenosti nám mohou posloužit jako představa reálného použití ebXML zpráv.

V praxi se například vyskytly zprávy uložené v jiné znakové sadě, než UTF-8, které ovšem tuto skutečnost neměly uvedenu ve své hlavičce. Neznalý čtenář, který by se je pokusil přečíst a zpracovat v kódování UTF-8, jak říká doporučení konsorcia W3C, by neuspěl. Některá pořadí znaků jiných kódování totiž nelze ve víceznakovém kódování UTF-8 přečíst.

Zatím se také nevyskytla zpráva, kde by nějaký XML tag obsahoval zároveň text a zároveň i další tagy. Tento fakt lze brát za, bohužel nepsaný, standard. Lze jej výhodně využít pro lepší návrh šablon Dispatchera s minimální ztrátou funkcionality.

2.3 Standard Inhouse

Standard Inhouse je největší potíží celého zpracování. Nejedná se totiž o standard – odesilatel a příjemce se dohodnou na libovolném formátu, ve kterém si data předají.

Pěkným příkladem standardu Inhouse mohou být zprávy německého standardu Der Verband der Automobilindustrie (VDA). Jednotlivé segmenty jsou umístěny za sebe a jsou právě 512 byte dlouhé. První tři znaky každého segmentu obsahují číslo identifikující typ segmentu. Stejně jako v případě standardu EDIFACT se mohou segmenty ve skupinách opakovat.

Jiným příkladem standardu Inhouse může být soubor /etc/passwd standardního unixového systému. Každý segment je zde zapsán na samostatném řádku, skládá se právě z šesti elementů oddělených znakem dvojtečka.

Dalším příkladem může být soubor s hodnotami zapsanými ve formátu CSV (Comma Separated Value). Bohužel, někteří odesilatelé či příjemci používají jako oddělovač elementů místo čárek středníky, občas také používají otazník jako zprošťující znak. Běžně se přitom hodnoty obsahující čárku uzavírají do uvozovek, uvozovky v hodnotě se zdvojují. Zpracování zpráv ve formátu CSV a podobném proto zatím odložme. Až si praktické použití Dispatchera vyžádá zpracování konkrétní implementace těchto zpráv, nebude obtížné dopsat kód pro jejich zpracování.

3 Formát šablon

3.1 Obecná struktura

Pokusme se nyní navrhnout formát šablon zpráv tak, aby splnil základní požadavky:

1. Byl schopen popsat strukturu libovolné zprávy
2. Byl co nejvíce podobný pro všechny standardy zpráv
3. Byl lehce čitelný a zapisovatelný člověkem

Jelikož největší množství zpracovávaných zpráv je standardu EDIFACT, bylo by dobré, aby se obecná struktura šablon podobala formátu EDIFACT zpráv. Při zápisu a pozdějším čtení tato podobnost jistě přijde vhod.

První myšlenkou, jak zapsat šablony, by mohl být prostý regulární výraz. Regulární výraz by byl schopen popsat libovolný text – jenže ve zprávách často potřebujeme více funkcionality, než pouhé rozpoznávání textu. Potřebujeme schopnost označit určitý kus textu jako informaci, kterou požadujeme v nálepce. Navíc regulární výraz pro celou zprávu by byl velice dlouhý, a tudíž velice obtížně udržovatelný a navíc téměř nečitelný člověkem.

Pokusme se nyní sestavit strukturu šablony tak, aby byla schopna popsat strukturu zprávy a popsat také umístění hledaných informací ve zprávě.

3.2 Převod zpráv na seznam segmentů

Nejprve je dobré si všimnout, že každou zprávu lze přepsat na posloupnost segmentů. Standard EDIFACT už přímo posloupností segmentů je.

Vzhledem k tomu, že tagy ve zprávách standardu ebXML většinou obsahují buď pouze jiné tagy a nebo pouze text, můžeme převést ebXML zprávy na posloupnost segmentů následovně:

- Obsahuje-li tag pouze text a žádné jiné tagy, jeho koncový tag zrušme a udělejme z něj jednoduchý segment.
- Obsahuje-li tag pouze jiné tagy a žádný text, rozdělme jej na dva segmenty – jeden počáteční a jeden koncový. Tyto dva segmenty budou hrát roli skupiny segmentů.
- Pokud tag obsahuje jiné tagy i text, rozdělme jej na dva segmenty a veškerý volný obsah přidělme prvnímu segmentu. Tato situace v praxi nenastává, proto její řešení můžeme zvolit libovolně.

Tímto postupem se nám podaří rozbít stromovitou strukturu ebXML zpráv a převést ji na lineární pořadí segmentů. Přitom ztratíme jedinou informaci – přesné rozmístění textu mezi vnořenými tagy v případě, že tag obsahoval i text i jiné tagy. Vzhledem k tomu, že tato situace v praxi nenastává, nejedná se o velký problém.

Ve zprávách standardu ebXML jsme mohli na dva segmenty převést všechny tagy – každý někde začíná a končí. Neztratili bychom poté informaci o přesném umístění vnořených dat v tagu. Jenže poté by se šablona pro standard ebXML velice lišila od šablony pro standard EDIFACT. V šablonách standardu EDIFACT se totiž konce segmentů

nevyskytují. Bylo by také velice nepraktické uvádět v šabloně za každým segmentem jeho konec. Stačí uvádět jej pouze za těmi segmenty, které slouží jako obálka jiných segmentů. Tyto segmenty jsou tedy jistou obdobou skupin segmentů ve standardu EDIFACT.

Se zprávami standardu inhouse je tradičně potíž. Doposud vždy byly tyto zprávy tvořeny segmenty seřazenými lineárně za sebou. Jen jednotlivé separátory segmentů a jejich částí se lišily. Někdy je na konci segmentů daný koncový znak, jindy (jako v případě zpráv VDA) má každý segment předem danou délku. Informace se pak v segmentu nacházejí na předem dohodnutých pozicích.

Díky tomu je ale možné považovat inhouse zprávu za lineární seznam segmentů.

3.3 Šablona jako popis zprávy

Každou zprávu tedy můžeme považovat za lineární seznam segmentů. I šablona proto bude tvořena seznamem segmentů. Nechť má každý segment šablony schopnost porovnat se se segmentem zprávy a říct, jestli mu tento segment zprávy odpovídá. Pokud bychom nebyli schopni zprávy rozdělit na jednotlivé segmenty (jako v případě zpráv standardu inhouse), nechť také segment šablony obsahuje informace potřebné pro vybrání segmentu ze zprávy. V šabloně musí být také uveden standard zprávy, aby byla jasná podoba segmentů šablony. Pak už by v šabloně chyběla jediná informace – které segmenty spolu tvoří skupiny, ve kterých se mohou opakovat. Přidejme tedy do šablony ještě speciální druh segmentů, které určují začátek a konec skupin. Tyto skupiny se pak mohou libovolně opakovat.

Konkrétní podobu segmentů šablony zatím nechme na později, blíže o tom odstavec 3.6: Popis segmentů šablony, strana 14.

Jak poznat standard zprávy, abychom ji mohli rozdělit na segmenty? Nabízí se jednoduchý algoritmus:

1. Začíná-li zpráva EDIFACT segmenty (UNA) UNB (UNG) UNH a končí segmenty UNT (UNE) UNZ podle normy ISO 9735, jedná se o standard EDIFACT. Segmenty v závorkách jsou nepovinné.
2. Je-li zpráva well-formed XML (až na kódování), jedná se o zprávu standardu ebXML.
3. Jinak se jedná o zprávu standardu inhouse.

Jenže tento postup není nejlepší – co kdyby inhouse zpráva také začínala segmenty, kterými jinak začíná EDIFACT zpráva? Tato možnost by sice neměla nastat (byla by to velká nezodpovědnost autora inhouse zprávy), ale i tak lze problém vyřešit lépe.

Nechť je v každé šabloně uveden standard zprávy, pro který je napsána. Při zpracování šablony po přečtení tohoto standardu zkontrolujeme, jestli zpráva splňuje výše uvedené podmínky. Pokud ano, pokračujeme ve zpracování šablony. Díky tomu se obecný požadavek na strukturu zprávy stává nutnou, nikoliv však postačující podmínkou pro to, aby odpovídala zpráva šabloně.

Nyní již můžeme jednoduše rozhodnout, jestli zpráva odpovídá šabloně. Stačí udržovat si seznam pozic v šabloně, porovnávat segmenty na těchto pozicích se segmenty zprávy a pokud v šabloně dojdeme až na konec, odpovídá zpráva šabloně. Zprávu porovnáváme se všemi dostupnými šablonami v daném pořadí, až najdeme první, které zpráva odpovídá. Pokud žádnou šablonu, které by zpráva odpovídala, nenajdeme, odložíme zprávu stranou pro ruční zpracování. Takovou zprávu nejsme schopni korektně identifikovat.

Některé segmenty zprávy ovšem mohou být nepovinné. Ve zprávě se vyskytovat mohou, ale nemusí. Standard EDIFACT například definuje u každého segmentu, jestli se tento segment ve zprávě vyskytovat musí, nebo ne. Proto i u každého segmentu šablony je nutné uvést jeho povinnost.

Aby nebylo nutné vypisovat všechny, povinné i nepovinné, segmenty zprávy do šablony, je dobré přidat ještě jedno pravidlo. Pokud se ve zprávě vyskytuje segment, na který zrovna v šabloně nečekáme, tento segment přeskočme. Chceme-li tedy například zkontrolovat v EDIFACT zprávě, že se v ní vykytuje segment UNG, stačí do šablony zapsat pouze tento segment. Není nutné vyjmenovávat všechny povinně předcházející segmenty.

3.4 Možná rozšíření

Celková struktura šablony se podobá regulárním výrazům. Pokud dojdeme na konec regulárního výrazu, porovnávaný text odpovídal výrazu. Stejně tak pokud dojdeme na konec šablony, odpovídala zpráva šabloně. Pouze místo písmen zde máme segmenty, které lze ale porovnávat se segmenty zprávy stejně jako písmena regulárního výrazu s písmeny rozpoznávaného textu. Šablona je jakýmsi „regulárním výrazem zapsaným na výšku“.

Podobnost s regulárními výrazy nám také dává nápady na možné rozšíření šablon v budoucnu. Například můžeme přidat jakýsi „univerzální segment“, kterému by odpovídal libovolný segment zprávy. Tím bychom mohli v šabloně říct, že chceme „pátý segment zprávy“. Nebo bychom mohli přidat rovnou množinu segmentů – segment zprávy by odpovídal této množině, pokud by odpovídal alespoň jednomu segmentu z množiny. Nápadů na rozšíření je mnoho, až podrobné testování ukáže, která funkcionalita by přišla vhod.

3.5 Šablona jako popis umístění hledaných informací ve zprávě

Druhou požadovanou schopností šablony je popsat umístění hledaných informací ve zprávě.

Použijeme výše zmíněnou lineární strukturu segmentů. Do každého segmentu připíšeme informaci, kde přesně se v něm nachází hledané informace.

Každou hledanou informaci označme nějakým slovem, pod kterým bude zapsána v nálepce. Můžeme pak vytvořit jakousi mapu, indexovaný seznam, hodnot vybraných ze zprávy. Označme tuto mapu jako pole hodnot _PUBLIC. Jakákoli hodnota, která se vyskytne v tomto poli, se na konci práce Dispatchera vypíše do nálepky. Jedná se vlastně o pole veřejných proměnných.

Pro jednoduchost nechť názvy prvků v poli nezávisí na velikosti písmen. Proměnná _PUBLIC[Prom] bude mít tedy stejnou hodnotu jako proměnná _PUBLIC[prOm].

Znak podtržítka na začátku nechť je společný pro všechna klíčová slova, která Dispatcher v šabloně rozeznává.

V praxi se nám ale bude hodit ještě jedno pole – pole interních proměnných. Do těchto interních proměnných si můžeme vynést některé hodnoty ze zprávy, detailněji je prozkoumat, a teprve pokud splňují určitá pravidla, překopírovat je do veřejného pole _PUBLIC. Toto interní pole označme _PRIVATE.

Jak tedy označíme data, které chceme zkopírovat do některého pole proměnných? Do šablony segmentu na místo, kde tato data nacházejí, pouze napíšeme přesný název proměnné v poli, kam chceme tuto hodnotu vykopírovat. Například _PUBLIC[var] – ve výsledné nálepce se pak objeví proměnná „var“, která bude mít jako svůj obsah data ze zprávy. Název proměnné v poli není nutné uzavírat do uvozovek, protože je již ohraničen závorkami. Příklady zápisu pro jednotlivé standardy viz dále v odstavci 3.6: Popis segmentů šablony, strana 14.

3.6 Popis segmentů šablony

Pokusme se nyní navrhnout strukturu segmentů šablony. Tato struktura musí být schopna popsat libovolnou podobu segmentů zpráv jednotlivých standardů a popsat také, kde se v segmentu nachází požadované informace.

Bohužel, struktura nutně musí být různá pro všechny tři druhy zpráv. Ve zprávách standardu inhouse například potřebujeme znát identifikaci konce segmentu, kterou u zpráv ebXML a EDIFACT nepotřebujeme. Šablony standardu EDIFACT zase musí obsahovat číslo verze EDIFACT zprávy, abychom byli schopni určit, které znaky jsou separátory. Přesto by bylo dobré, aby se obecná struktura segmentů šablony co nejvíce podobala segmentům zpráv standardu EDIFACT.

U segmentů všech zpráv potřebujeme znát informaci, jestli se tento segment vyskytuje ve zprávě povinně, nebo nepovinně. Proto nechť každý segment šablony začíná jedním znakem následovaným mezerou:

- M (mandatory) pro povinné segmenty
- C (conditional) pro nepovinné segmenty

Za touto mezerou pak následuje samotný popis segmentu.

3.6.1 Standard EDIFACT

Každý segment zprávy standardu EDIFACT se skládá z informací oddělených předem danými oddělovači. Tyto oddělovače (separátory) mohou být na začátku zprávu přetíženy. Předpokládejme proto, že prvním krokem zpracování EDIFACT zprávy bude nahrazení všech separátorů ve zprávě za výchozí hodnoty. Standardní separátory jsou dány normou ISO 9735. Tyto separátory obsahuje také Tabulka 1: Výchozí EDIFACT separátory, strana 8.

Jistý problém je s výchozím separátorem opakujících se elementů. Tento znak (implicitně hvězdička) je do verze 3 normálním znakem, od verze 4 nabývá významu zmiňovaného separátoru. Proto před samotným zpracováním je pro nahrazení separátorů za výchozí hodnoty nutné přečíst ze zprávy její verzi. Pokud je tato verze ≥ 4, je nutné i znak hvězdička zařadit mezi separátory.

V každém segmentu se tedy nachází vlastní data prokládaná nějakým množstvím různých separátorů. Pokud mezi separátory není vložen žádný text, mohou být tyto separátory vypuštěny bez změny významu segmentu (viz kapitolu 7: Komprese normy ISO 9735).

V dalším textu nebereme v potaz separátory, jejichž význam je zrušen pomocí zprošťujícího znaku otazník. Stejně tak uvažujeme pouze výchozí EDIFACT separátory – příslušné znaky samozřejmě mohou být v nepovinném segmentu UNA změněny.

Všimněme si, že separátory lze seřadit podle významu. Méně významné separátory se ve zprávě vkládají mezi více významné separátory. Nejdůležitějším separátorem je konec segmentu, který se vyskytuje v každém segmentu právě jedou. Mezi začátkem a koncem segmentu může být vloženo několik oddělovačů elementů, tedy znaků plus. Pokud se jedná o elementy složené, jsou části složeného elementu odděleny znakem dvojtečka. Pokud se jedná o oddělení opakujících se elementů, je mezi nimi znak hvězdička. Tabulka 2: Priorita separátorů EDIFACT obsahuje ucelený přehled.

Priorita	Znak	Význam
1.	' (apostrof)	Konec segmentu
2.	+ (plus)	Separátor elementů
3.	: (dvojtečka)	Separátor prvků složeného elementu
4.	* (hvězdička)	Separátor opakujících se elementů (verze 4)

Tabulka 2: Priorita separátorů EDIFACT

Mezi separátory se v segmentu šablony mohou nacházet následující možnosti:

- Nic – v segmentu zprávy mohou být libovolná data až do dalšího separátoru, který následuje v šabloně
- Označení proměnné v jednom z polí _PUBLIC či _PRIVATE – vykopíruj ze segmentu zprávy data až do následujícího libovolného separátoru, ty umísti do dané proměnné. Poté se přesun až k takovému separátoru, jaký následuje v šabloně.
- Jiné znaky – porovnej tyto znaky s textem ve zprávě až do následujícího separátoru. Pokud jsou různé, segmenty si neodpovídají. Poté se ve zprávě přesuň až k dalšímu takovému separátoru, jaký je uveden v šabloně.

V případě, že bylo dosaženo konce segmentu zprávy dříve než konce segmentu šablony, odpovídají si segmenty právě tehdy když v segmentu šablony nebyly už žádné další volné znaky k porovnání. Vlastní porovnávání segmentů shrňme do následujícího algoritmu, který postupně prochází znaky segmentu šablony:

- Je-li v šabloně znak různý od separátorů a znaku _, kterým začíná označení pole proměnných, porovnej jej se znakem ve zprávě. Jsou-li různé, segmenty si neodpovídají.
- Je-li v šabloně separátor, čti zprávu tak dlouho, až najdeš stejný separátor. Pokud jsi našel konec segmentu zprávy, segmenty si odpovídají právě když v šabloně už nejsou žádné další znaky k porovnání.
- Je-li v šabloně proměnná z pole _PUBLIC či _PRIVATE, čti zprávu až do prvního libovolného separátoru. Tento úsek ulož do dané proměnné. Poté čti až do stejného separátoru, jaký byl v šabloně. Pokud jsi našel konec segmentu zprávy, segmenty si odpovídají právě když v šabloně už nejsou žádné další znaky k porovnání.

Tento algoritmus samozřejmě také musí vzít v potaz znaky, jejich význam byl zrušen zprošťujícím znakem otazník. Také musíme odlišit významy znaku podtržítko. Samotné podtržítko značí proměnnou z polí _PUBLIC či _PRIVATE. Pokud požadujeme podtržítko ve zprávě, stačí před něj přidat zprošťující znak otazník.

3.6.1.1 Příklad

Uveďme nyní příklad segmentu zprávy a šablony pro standard EDIFACT. Jedná se o minimální segment UNB, viz normu ISO 9735, příloha B.

Segment zprávy:

```
UNB+UNOD:3+7XRRD+HDLLD+940101:0950+1'
```

Segment šablony:

```
M UNB+_PUBLIC[charset]+7XRRD++_PUBLIC[date]:_PUBLIC[time]
++_PUBLIC[password]'
```

Segment šablony říká, že tento segment je ve zprávě povinný. Tyto dva segmenty si zřejmě odpovídají – jediným požadavkem na obsah segmentu je zde třetí element (7XRRD), což je kód odesilatele. Ten je stejný. Zřejmě nás zde tedy zajímají všechny zprávy od určitého odesilatele.

Poté, co si segmenty odpovídají, dojde k přenesení informací (spuštění) segmentu s následujícím výsledkem:

_PUBLIC[charset] = „`UNOD`“

_PUBLIC[date] = „`940101`“

_PUBLIC[time] = „`0950`“

_PUBLIC[password] = „“

3.6.2 Standard ebXML

Ve zprávách ebXML se nacházejí dva typy segmentů. Prvním jsou datové segmenty, které obsahují nějaká data v obsahu Druhým jsou pak segmenty sloužící jako obálka. Pro zpracování v šabloně už ale tento rozdíl můžeme zanedbat. Název koncových segmentů obálek pouze začíná znakem lomítka. Pro tento převod viz část 3.2: Převod zpráv na seznam segmentů, strana 11.

Každý segment může obsahovat jednak nějaké atributy, jednak určitý obsah (kromě koncového segmentu obálky). Atributy označme jejich jmény. Pro obsah tagu použijme speciálního jména _content.

Jako alternativní možnost se zde nabízí přepsání všech atributů na vnořené tagy, které ponesou hodnotu atributu ve svém obsahu. Tím by se ale zbytečně prodloužila délka šablon a obecně by mohlo dojít ke konfliktu názvů atributů a vnořených segmentů.

Každý segment ebXML šablony nechť začíná názvem segmentu (tagu) a následuje seznam atributů s přiřazením či porovnáním hodnot:

- Pro přiřazení použijme symbol rovná se =. Použije se pro vykopírování dat z atributu do proměnné
- Pro porovnání použijme symbol dvou rovná se ==. Použije se pro porovnání atributu s hodnotou proměnné či konstantním řetězcem.

Tím jsme schopni klást libovolné požadavky na obsah atributů či obsahu, stejně jako požadavky na vykopírování hodnot ze šablony.

Pro praktické zpracování ebXML segmentů je nutné si uvědomit, že názvy tagů v XML zprávách závisí na velikosti písmen. Také název a atributy segmentu šablony proto rozlišují velikost písmen.

3.6.2.1 Příklad

Uveďme delší fiktivní příklad XML tagu zprávy. Příklad pravděpodobně nesplňuje sémantické podmínky ebXML zprávy, jako příklad XML tagu nám ale poslouží dobře. Uveďme tradiční příklad tagu reprezentujícího osobu:

Tagy ebXML zprávy:

```
<osoba pohlaví="žena">
      <jméno typ="současné">Pavla Veselá</jméno>
      <jméno typ="rodné">Pavla Svobodná</jméno>
      <bydliště>U pštrosa 4</bydliště>
</osoba>
```

Segmenty šablony:

```
M osoba _PUBLIC[sex]=pohlaví
M jméno typ=="současné" _PUBLIC[name_actual]=_content
C jméno typ=="rodné" _PUBLIC[name_birth]=_content
M bydliště _PUBLIC[residence_permanent]=trvalé
M /osoba
```

Ze šablony můžeme vidět, že sice celý tag osoba je povinný, rodné jméno je v něm ale uvedeno jako nepovinná informace. Pokud se ovšem ve zprávě vyskytne, pak si jej přejeme vykopírovat do nálepky. U tagu bydliště požadujeme atribut „trvalé“. Tento atribut ovšem ve zprávě uveden není – proto se vyhodnotí jako prázdný.

Výsledné proměnné budou naplněny takto:

_PUBLIC[sex] = „žena“

_PUBLIC[name_actual] = „Pavla Veselá“

_PUBLIC[name_birth] = „Pavla Svobodná“

_PUBLIC[residence_permanent] = „“

Formát segmentu šablony ebXML je také možné rozšířit. Například lze přidat symbol „!=“ značící, že atribut se nesmí rovnat dané hodnotě. Dále můžeme umožnit porovnání pouze podle několika prvních znaků. Která rozšíření se ukážou jako rozumná ukáže opět až fáze reálného testování.

Je dobré si také povšimnout, že příklad šablony není dokonalý. Pokud by například ve zprávě došlo k záměně pořadí tagů „jméno“, nedostali bychom při zpracování hodnoty z tagu „jméno“ s atributem „typ=rodné“. K uspokojivému vyřešení tohoto problému budeme potřebovat podmíněné příkazy nebo označení pro skupiny segmentů. Také nám přijde vhod pole interních proměnných _PRIVATE. Pro popis příkazů viz část 3.7.2: Příkazy, strana 22. Pro označení skupin segmentů viz část 3.7.1: Označení skupin segmentů, strana 20.

3.6.3 Standard inhouse

Se standardem inhouse je tradičně potíž. Jak popsat strukturu libovolného textu jinak, než regulárním výrazem? Jak případně do regulárního výrazu přidat informace o tom, které části textu jsou pro nás významné? Je vůbec možné zapsat regulární výraz pro jeden segment, když například znaky, sloužící jako separátory a zprošťující znaky, mohou být

uveden na začátku zprávy podobně jako v případě standardu EDIFACT? Regulární výraz pro celou zprávu by byl samozřejmě velice dlouhý a složitý.

Zavrhněme proto nápad s regulárními výrazy. Pokusme se o systém kombinující prvky ze šablon standardu EDIFACT a standardu ebXML.

Všechny doposud zpracovávané zprávy inhouse se skládaly ze segmentů lineárně zapsaných za sebou. Jakým způsobem je možné signalizovat konec segmentu? Buď je možné na konci uvést nějaký koncový znak, nebo musí být délka segmentu předem známa. Proto rozdělme segmenty standardu inhouse na dva podtypy:

- variabilní segmenty – jejich délky jsou různé, ale vždy končí nějakým koncovým znakem
- konstantní segmenty – jejich délky jsou předem dohodnuté

Do hlavičky inhouse šablony je třeba přidat řádek, který identifikuje konec segmentu a typ inhouse zprávy. Formát tohoto řádku zvolme následovně:

```
INHOUSE_FORMAT <typ> <identifikace>
```

Kde

```
<typ> ~ C|V
```

C pro konstantní segmenty, V pro variabilní.

```
<identifikace> ~ <idConst> | <idVar>
<idVar> ~ _length==“délka“ | λ
```

_length je nepovinný parametr udávající délku všech segmentů

```
<idConst> ~ právě 4 znaky udávající separátory
```

Jako znaky reprezentující separátory lze použít následující 4 speciální spojení:

- `\n` – konec řádku
- `\t` – tabulátor
- `\\` – zpětné lomítko
- `\x` – Symbol není použit, tento separátor se ve zprávě nevyskytuje

3.6.3.1 Inhouse s variabilními segmenty

Standard inhouse s variabilními segmenty se podobá zprávám standardu EDIFACT. Proto v každé šabloně definujeme raději hned několik separátorů, podobně jako jsou v EDIFACT zprávách. Tyto separátory budou seřazeny podle priority stejně jako v případě EDIFACT zpráv. Pokud bude ve zprávě použito méně separátorů než definujeme v šabloně, stačí tyto separátory nastavit na předem dohodnuté „mrtvé“ znaky. Dispatcher tyto znaky poté nebude považovat za separátory.

EDIFACT do verze 3 (která se doposud ve světě používá v naprosté většině EDIFACT zpráv) vystačí se třemi separátory. Proto i v případě variabilních inhouse zpráv používejme maximálně 3 separátory. Jestli je tento počet nedostatečný ukáže až testování.

Pro vlastní zpracování inhouse segmentů můžeme s výhodou využít kód již napsaný pro zpracování EDIFACT segmentů. Stačí jen zaměnit separátory a nastavit verzi EDIFACT na 3. Je zde ale pár rozdílů. Segment zprávy EDIFACT vždy musí končit znakem pro konec segmentu (implicitně apostrof), zato segment zprávy inhouse tímto znakem končit nemusí. Ve zprávě inhouse bychom také neměli vykopírovat při určitém požadavku vždy hodnotu do prvního libovolného separátoru, ale měli bychom vykopírovat vždy celý obsah až do

následujícího separátoru daného v šabloně. Není zde totiž norma zprávy, která by zaručila počet po sobě následujících prvků složeného elementu v segmentu.

Spoléhat na podobnost s EDIFACT zprávami není tak špatný nápad, jak se může zdát. Naprostá většina návrhářů inhouse zpráv má určitě zkušenosti se zprávami standardu EDIFACT, a k těmto zkušenostem bude přihlížet.

3.6.3.2 Inhouse s konstantními segmenty

Schopnost zpracovávat zprávy inhouse s konstantními segmenty potřebujeme už kvůli zprávám standardu VDA. Pro větší obecnost je dobré udržet možnost různých délek jednotlivých segmentů, pro jednodušší psaní šablony pak možnost zadat jednu délku pro všechny segmenty. Tato délka nechť je zapsaná v hlavičce šablony.

Pro identifikaci jednotlivých dat můžeme použít umístění zadané rozdílem pozic v segmentu. Důležité informace se mohou nacházet pouze na pozicích A až B.

3.6.3.3 Příklad

Oba dva výše zmíněné případy se velice liší. Proto uvedeme pro standard inhouse příklady dva – jeden pro konstantní inhouse, druhý pro variabilní. V hlavičce celé šablony nechť se nachází jeden z následujících řádků:

```
INHOUSE_FORMAT C _length=“délka“
INHOUSE_FORMAT V :+?'
```

Písmeno C označuje konstantní typ inhouse zprávy, písmeno V pak variabilní. Parametr _length je nepovinný, slouží k zadání délky segmentu pro všechny segmenty na jednom místě. Výpis separátorů u variabilního inhouse pak je povinný, bez nich bychom nemohli zprávu zpracovat.

Fiktivní segment konstantní zprávy standardu inhouse:

```
PRIJEMCE OBJEDNAVKA200805051950
```

Příslušný segment šablony:

```
M _PUBLIC[reciever]=1-10 11-20==“OBJEDNAVKA“
_PUBLIC[date]=21-28 _PUBLIC[time]=29-32
```

Zajímá nás tedy zřejmě pouze segment, který má na pozicích 11-20 řetězec OBJEDNAVKA, znaky se číslují od 1. Výsledné proměnné budou naplněny takto:

_PUBLIC[reciever] = „PRIJEMCE “

_PUBLIC[date] = „20080505“

_PUBLIC[time] = „1950“

Je dobré si povšimnout, že hodnota proměnné reciever v poli _PUBLIC končí dvěma mezerami Tyto mezery byly ve zprávě na daných pozicích uvedeny.

Předpokládejme nyní, že separátory variabilní inhouse zprávy byly v hlavičce šablony nastaveny takto:

- `,` – čárka jako separátor prvků složeného elementu
- `:` – dvojtečka jako separátor elementů
- `\\` – zpětné lomítko jako zprošťující znak
- `\n` – nový řádek jako konec segmentu

Fiktivní segment variabilní zprávy standardu inhouse:

```
users:x:100:pepa,karel\n
```

Příslušný segment šablony:

```
M users++_PUBLIC[gid]+_PUBLIC[members]\n
```

Segment šablony reprezentuje řádek standardního unixového souboru /etc/group. Tento soubor se nám hodí jako pěkný příklad variabilní inhouse zprávy.

Výsledné proměnné:

_PUBLIC[gid] = „`100`“

_PUBLIC[] = „`pepa:karel`“

Zde je dobré si povšimnout, že došlo k nahrazení separátorů za výchozí EDIFACT hodnoty. Díky tomu je následné zpracování průvodky jednodušší. Kdyby se ve zprávě vyskytl jeden z výchozích separátorů EDIFACT jako normální znak, je nutné před něj přidat znak otazník jako výchozí zprošťující znak standardu EDIFACT. Stejně tak znaky otazník je nutné zdvojit.

Pokud dojde k zápisu nálepky do souboru, dojde k uzavření hodnoty proměnných do uvozovek. Uvozovky v hodnotě proměnné jsou poté zproštěny svého významu standardním zprošťujícím znakem \. Může se zdát, že dojde ke zbytečnému dvojímu zprošťování významu speciálních znaků v hodnotách proměnných – poprvé při standardizaci separátorů na výchozí EDIFACT separátory, podruhé při zápisu do průvodky. Každý z těchto převodů má ale svůj význam a není rozumné jej vynechat.

Popsaná struktura inhouse šablon samozřejmě není všemocná. Chybí nám například možnost využít více než tři separátory, využít jako separátory sekvence znaků, nebo nějaká schopnost reprezentace hierarchické struktury přenášených segmentů, jak jsou zapsány například zprávy XML. Popsaná struktura je ovšem schopna popsat doposud zpracovávané zprávy standardu inhouse. Pokud se v budoucnu objeví formát, který nelze těmito šablonami popsat, bude nutné dopsat nový modul, který tyto zprávy zpracuje.

3.7 Další prvky šablony

Máme tedy danou strukturu segmentů šablony, které jsou schopny popsat libovolný jeden segment zpráv všech standardů. Nyní je potřeba přidat do šablony další druhy segmentů, které budou schopny popsat zbylé informace nutné ke zpracování zprávy. Některé z těchto „speciálních segmentů“ již byly představeny.

3.7.1 Označení skupin segmentů

Ve zprávách standardu EDIFACT se segmenty nacházejí ve skupinách. Každá z těchto skupin segmentů má typem zprávy danou povinnost a maximální počet opakování. Stejné skupiny se nacházejí například také ve zprávách standardu VDA. Ve zprávách standardu ebXML nám také skupiny přijdou vhod. Musíme tedy přidat do šablony informace o skupinách segmentů a jejich povinnostech. Maximální počet opakování můžeme zanedbat – předpokládejme zkrátka, že každá skupina se může opakovat libovolně krát.

Pro začátek skupiny segmentů použijme následující zápis:

```
M {
```

Pro konec skupiny segmentů použijme následující:

```
}
```

První písmeno u začátku grupy segmentů označuje povinnost celé skupiny:

- M (mandatory) pro povinné skupiny
- C (conditional) pro nepovinné skupiny

Začátek skupiny segmentů tedy používá stejný zápis povinnosti jako normální segment. Segmenty ve skupině se mohou libovolně krát opakovat.

Jaký význam mají skupiny segmentů ve zprávách standardu ebXML? Přesně stejný, jako ve zprávách ostatních standardů. Uzavření segmentu reprezentujícího pouze začátek nebo konec obálkového segmentu nemá význam, vždy je potřeba přidat i příslušný druhý segment. Pokud ale do skupiny uzavřeme některé datové segmenty nebo celý obálkový segment, budou se tyto segmenty moci libovolně opakovat. Díky tomu jsme schopni zpracovat libovolné množství opakujících se segmentů.

3.7.1.1 Příklad

Jako příklad uveďme několik segmentů zprávy EDIFACT typu ORDERS. Chceme zpracovat všechny segmenty NAD stran, které jsou do zprávy zapojeny. Umístění (segment LOC) ani finanční instituce (segment FII) nás u těchto stran nezajímají.

Segmenty zprávy:

```
NAD+BY+8589000009401::9'
NAD+DP+8589000009418::9'
NAD+IV+8589000009494::9' Příkazy
```

Segmenty šablony:

```
C {
M NAD+_PUBLIC[type]+_PUBLIC[id]+PUBLIC[name]'
}
```

V šabloně vidíme, že segmenty NAD se nachází v nepovinné grupě, ale v této grupě jsou již povinné. V případě grupy o jednom segmentu toto pořadí povinností nemá vliv, pokud by ale bylo v grupě segmentů více, hraje pořadí velkou roli.

Šablona sice získá všechny požadované informace ze segmentů, je ale nutné si uvědomit, že ve výsledné nálepce se stejně neobjeví. Protože mají proměnné pro všechny segmenty NAD stejné jméno (jsou zapsány jedním segmentem šablony v grupě), dojde k přepsání hodnot v proměnných.

Jediným řešením je automatické číslování segmentů v grupě. Zaveďme proto speciální znak, který bude při zpracování nahrazován za pořadové číslo spuštění segmentu. Za tento znak zvolme #. Abychom získali všechny zapojené strany ze segmentů NAD do nálepky, musíme segment šablony upravit následovně:

```
C {
M NAD+_PUBLIC[type_#]+_PUBLIC[id_#]+PUBLIC[name_#]'
}
```

Výsledné proměnné pak budou naplněny takto:

_PUBLIC[type_01] = „BY“

_PUBLIC[id_01] = „8589000009401“

_PUBLIC[name_01] = „“

_PUBLIC[type_02] = „DP“

_PUBLIC[id_02] = „8589000009418“

_PUBLIC[name_02] = „“

_PUBLIC[type_03] = „IV“

_PUBLIC[id_03] = „8589000009494“

_PUBLIC[name_03] = „“

3.7.2 Příkazy

Přidejme nyní do šablony ještě speciální řádky, které budou reprezentovat příkazy. Pro začátek nám bude stačit pouze jeden příkaz, reprezentovaný jednoduchou gramatikou:

```
_if (<podmínka>) then <příkaz>
```

kde podmínka má následující gramatiku:

```
<podmínka> ~ <atom> <symbol> <atom>
<atom> ~ řetězec | proměnná
<symbol> ~ == | !=
```

Příkaz má pak následující gramatiku:

```
<příkaz> ~ proměnná = <atom> | _return
```

Proměnná zde reprezentuje jednu z proměnných v poli _PUBLIC nebo _PRIVATE. Řetězec reprezentuje libovolný znakový řetězec uzavřený v uvozovkách, uvozovky v řetězci jsou zproštěny svého významu znakem zpětné lomítko. Symbol == slouží pro test shody, != pro test neshody, = pro přiřazení. Příkaz _return způsobí okamžité ukončení zpracování šablony takové, že zpráva neodpovídala šabloně.

Tento příkaz nám umožní nejprve vykopírovat ze segmentu zprávy určité hodnoty do proměnných v poli _PRIVATE, zde udělat jejich test a poté je teprve přesunout do pole _PUBLIC.

Také u příkazu _if využijeme symbol #, který je automaticky nahrazován za pořadové číslo spuštění segmentu. Díky tomu můžeme používat příkaz _if i ve skupinách segmentů.

3.7.2.1 Příklad

Pěkným příkladem by mohla být například šablona pro zpracování segmentů NAD zprávy EDIFACT typu ORDERS.

Segment zprávy:

```
NAD+BY++ODESILATEL ID::::+++++EW4 34J'
```

Segmenty šablony:

```
M NAD+_PRIVATE[type]+_PUBLIC[id]+_PRIVATE[name]
_if (_PRIVATE[type]=="BY") then
_PUBLIC[buyer]=_PRIVATE[name]
_if (_PRIVATE[type]=="PE") then
_PUBLIC[payee]=_PRIVATE[name]
```

Tyto segmenty slouží k testu, jestli segment NAD identifikoval kupce nebo příjemce platby. Pokud bychom výše uvedené segmenty uzavřely do skupiny, mohli bychom zpracovat segmenty NAD uvedené v libovolném pořadí.

3.7.3 Komentáře

Poslední prvek, který nám v šabloně prokazatelně chybí, jsou komentáře. Zaveďme proto znak, který umožní označit celý řádek za komentář:

Začíná-li řádek znakem #, je považován za komentář.

Jednořádkové komentáře nám prozatím budou stačit. Také komentáře na koncích řádků nebudou zatím nutné. V budoucnu je samozřejmě možné tyto komentáře přidat.

3.8 Budoucí rozšíření

Je mnoho oblastí šablony, které je možné dále obohatit. Příkaz _if může být rozšířen o else větev, umožnit zpracování složitých podmínek či víceřádkových příkazů. Můžeme přidat další možnosti komentování řádků šablony či vymyslet lepší systém pro popis segmentů inhouse zpráv. Bohužel se ale nedá nyní odhadnout, která z těchto funkcionalit by byla užitečná, která by se nevyužívala či která by byla dokonce překážkou v provozu Dispatchera. Nezbývá proto nic jiného, než věnovat delší čas testování aplikace.

4 Princip práce Dispatchera

Podívejme se nyní na princip vlastní práce modulu Dispatcher.

Jak by se měl tento modul používat? Pro začátek předpokládejme jednoduchý princip. Po spuštění Dispatcher prozkoumá vstupní adresář se zprávami, zprávy jednu po druhé zpracuje a přesune do výstupního adresáře, kde je může využít další modul systému. Pokud by Dispatcher neuspěl se zpracováním nějaké zprávy, přesune tuto zprávu do chybového adresáře. Při zpracování každé zprávy postupně prochází šablonami, až najde jednu, která odpovídá zprávě. Protože na pořadí šablon záleží kolik informací se vykopíruje ze zprávy do nálepky, je nutné přesně stanovit pořadí zpracování šablon. Například obecná šablona, které odpovídá naprostá většina zpráv, musí být zařazena až po konkrétních šablonách, které mají velké požadavky na strukturu zprávy, ale získají z ní také velké množství informací. Ze stejného důvodu při výskytu syntaktické chyby v šabloně nestačí pouze vynechat tuto šablonu, ale je nutné ukončit zpracování této zprávy. Následující šablony totiž předpokládají, že zpráva neodpovídala všem předešlým šablonám. Nechť proto existuje soubor, ve kterém jsou šablony zapsány jedna za druhou.

Postup práce Dispatchera bychom mohli zapsat do následujícího algoritmu:

```
Načti konfigurační soubor.
Pro každou zprávu:
     Pro každou šablonu:
          Načti zprávu
          Načti šablonu
          Porovnej zprávu se šablonou (Parser):
               Pokud si odpovídají, ukonči hledání
               šablony.
               Pokud si neodpovídají, pokračuj další
               šablonou.
               Pokud je ve zprávě nebo šabloně
               syntaktická chyba, přesuň zprávu do
               chybového adresáře a skonči.
     Zapiš nálepku.
     Přesuň zprávu do výstupního adresáře.
     Pokračuj zpracováním další zprávy.
```

Načtení zprávy by stačilo provést pouze jednou pro všechny šablony. Bohužel před načtením šablony neznáme standard zprávy, a proto ji načíst nemůžeme. Standard zprávy by se sice dal určit heuristickou analýzou, tímto způsobem bychom ale ztratili užitečnou funkcionalitu. Více o tomto viz odstavec 3.3: Šablona jako popis zprávy (strana 12).

Podívejme se nyní na běh jednotlivých částí tak, jak byly uvedeny v algoritmu popsaném výše.

4.1 Načtení souboru s konfigurací

Pro spuštění vyžaduje Dispatcher právě jeden parametr – název souboru s konfigurací.

V konfiguračním souboru musí Dispatcher naleznout umístění všech pracovních adresářů. Také je zde ale možné inicializovat některé proměnné v polích _PUBLIC či _PRIVATE. Tím je možné zajistit, že se ve výsledné nálepce objeví určitá informace, bez

ohledu na fakt, jestli byla či nebyla nalezena příslušná šablona pro zprávu. V případě běhu několika modulů Dispatcher paralelně vedle sebe je zde možné zadat identifikační kód konkrétní instance Dispachera, například popis vstupního kanálu, nad kterým Dispatcher běží.

Po načtení konfiguračního souboru je provedena kontrola existence všech zadaných adresářů. V případě neexistence zadaného adresáře dojde k výpisu chyby na standardní výstup a okamžitému ukončení Dispatchera.

Konkrétní formát a struktura konfiguračního souboru jsou popsány v odstavci 5.3: Formát konfiguračního souboru, strana 33.

4.2 Načtení zprávy

Úkolem této části je načtení zprávy do interní struktury, její separace na seznam jednotlivých segmentů.

Předpokládejme, že z šablony jsme již obdrželi standard šablony. Ověříme, jestli zpráva splňuje nutná kritéria standardu:

- EDIFACT zpráva musí začínat segmenty (UNA) UNB (UNG) UNH podle normy ISO 9735, končit pak segmenty UNT (UNE) UNZ. V závorkách jsou uvedeny nepovinné segmenty.

  Celou EDIFACT zprávu dále musí být možné rozdělit podle separátorů segmentů (apostrof) na jednotlivé segmenty. Vzhledem k tomu, že neznáme složení jednotlivých segmentů, nemá tato podmínka praktický význam.

- ebXML zpráva musí být well-formed XML dokument. Jediný doposud zjištěný problém se týká kódování znaků. Pokud totiž předpokládáme, že je zpráva zapsána ve víceznakovém kódování UTF-8, ale přitom je zapsána například v kódování ISO-8859-2, nemusíme ji být schopni načíst. Více viz kapitolu 4.5: Znakové kódování zpráv, strana 30.

- Inhouse zpráva žádné podmínky na svou strukturu neklade.

Pokud zpráva tyto podmínky splňuje, pokračujeme v porovnávání zprávy se šablonou. Jinak zpráva šabloně neodpovídá.

Zprávy standardů EDIFACT a ebXML poté můžeme převést na lineární seznam segmentů (viz odstavec 3.2: Převod zpráv na seznam segmentů, strana 11). Problém je se zprávami standardu inhouse, u kterých potřebujeme informace z šablony (délku nebo koncový znak), abychom byli schopni získat segment.

Proto zprávy standardů EDIFACT a ebXML převedeme na seznam segmentů, se zprávou standardu inhouse zatím neuděláme nic. Segmenty z ní budeme získávat teprve při porovnávání s příslušnou šablonou, kdy budeme mít přístup k informacím obsaženým v šabloně.

4.3 Načtení šablony

Podívejme se nyní, jak vypadá načtení šablony a její finální podoba. Formát jednotlivých segmentů již byl popsán v Kapitole 3: Formát šablon, strana 11.

Každá šablona začíná hlavičkou. Prvním řádkem hlavičky je řádek, na kterém je uveden standard zprávy, pro který je šablona určena. Na velikosti písmen nezáleží. Jedná se o jedno z následujících slov:

- edifact pro zprávy standardu EDIFACT
- ebxml pro zprávy standardu ebXML
- inhouse pro zprávy standardu inhouse

Po načtení tohoto řádku v šabloně je zkontrolováno, že zpráva splňuje minimální požadavky na strukturu podle daného standardu. Pokud zpráva tuto strukturu nesplňuje, neodpovídá šabloně. Pro minimální požadavky jednotlivých standardů viz odstavec 4.2: Načtení zprávy, strana 25.

Dále zde může být uvedeno požadované kódování zprávy. Řádek s kódováním zprávy má následující podobu:

```
DEFAULT_ENCODING <kódování>
```

Kódování je jedna ze znakových sad nebo jejich aliasů schválených organizací IANA (viz http://www.iana.org/assignments/character-sets). O zpracování zpráv podle daného kódování viz odstavec 4.5: Znakové kódování zpráv, strana 30.

U zpráv standardu inhouse také následuje informace, jestli se jedná o variabilní inhouse zprávy, nebo konstantní. Tento řádek má následující formát:

```
INHOUSE_FORMAT <typ> <identifikace>
```

Typ je jedno z písmen C (pro konstantní inhouse) nebo V (pro variabilní inhouse).

Identifikace je pro variabilní inhouse povinná a má podobu právě 4 znaků reprezentujících po řadě EDIFACT separátory pro prvky složeného elementu (dvojtečka), jednotlivé elementy (plus), zprošťující znak (otazník) a konec segmentu (apostrof).

Identifikace pro konstantní inhouse je nepovinná. Může se zde nacházet parametr _length, který udává délku segmentů, pokud u jednotlivých segmentů šablony není uvedeno jinak.

Za hlavičkou šablony následuje jeden prázdný řádek.

Prázdné řádky a bílé znaky na začátcích a koncích řádků dále nemají význam. Řádek začínající znakem # je považován za komentář.

Jak bylo uvedeno v Kapitole 3: Formát šablon, ve vlastní šabloně následuje seznam řádků následujících typů:

- Segmenty šablony – jeden znak určující povinnost segmentu (M / C) následovaný popisem jednoho segmentu zprávy. Pro popis segmentů zpráv viz Kapitolu 3: Formát šablon, strana 11.
- Začátek skupiny segmentů – jeden znak určující povinnost celé skupiny následovaný znakem {.
- Konec skupiny segmentů – samotný znak } na řádku
- Příkaz – řádek začínající znakem _, doposud pouze příkaz _if.

Rozlišujeme tedy 4 typy segmentů. Celou šablonu převedeme na lineární seznam segmentů, z nichž každý bude právě jednoho z těchto 4 typů.

4.4 Porovnání a zpracování zprávy dle určité šablony (Parser)

Parser je vlastní jádro celého modulu Dispatcher. Jedná se o část, která je schopna porovnat zprávu (seznam segmentů zprávy) se šablonou (seznamem segmentů šablony) a jednoznačně určit, jestli tato zpráva odpovídá šabloně. Je také schopen extrahovat ze zprávy informace, které jsou v šabloně požadovány.

Rozdělme proto toto zpracování na dvě fáze:

- **Porovnáním** segmentů rozumějme pouze kontrolu, jestli segment zprávy splňuje strukturu požadovanou segmentem šablony. Zejména se jedná o kontrolu jména segmentu a znaků, které musí minimálně obsahovat.
- **Spuštěním** segmentu rozumějme extrakci informací ze segmentu zprávy do proměnných v polích _PUBLIC a _PRIVATE.

Kromě tohoto úkolu musí Parser zařídit ještě jednu věc. Vždy, když získává segment ze zprávy standardu inhouse, musí zprávě předat segment šablony, se kterým bude tento segment porovnávat. Jen díky tomu je schopna inhouse zpráva identifikovat konec dalšího segmentu a tento segment ze zprávy oddělit. Tato informace je zapsána pouze v šabloně.

Předpokládejme zatím, že každý segment šablony je schopen porovnat se se segmentem zprávy, tedy jednoznačně určit, jestli mu tento segment odpovídá, a také je schopen se podle tohoto segmentu spustit. O tuto činnost se pak jádro Parseru nemusí starat. Vlastní porovnávání a spouštění segmentů je blíže popsáno v odstavci 4.4.2: Porovnání segmentů zprávy se segmenty šablony, strana 28.

4.4.1 Princip běhu

V průběhu svého běhu si Parser udržuje seznam míst v šabloně, na kterých se potenciálně může nacházet aktuální pozice. Tato místa vnitřně reprezentujme jako jakési značky či tečky před určitými segmenty. Říkejme takto označeným segmentům **tečky** v šabloně. Těchto míst může být obecně více než jedno – viz dále. Jakmile je vyzvednut další segment ze zprávy, je každý takový označený segment šablony porovnán se segmentem vyzvednutým ze zprávy. Pokud segment zprávy odpovídá jednomu z označených segmentů, je označený segment spuštěn, tečka je posunuta za tento segment, a ostatní tečky jsou smazány. Pokud označený segment neodpovídá segmentu zprávy, dojde ke zrušení tečky. Na následující pozici v šabloně už se nemůžeme dostat. Jakmile se dostane tečka na konec šablony, zpráva odpovídá šabloně. Pokud všechny tečky v průběhu práce zaniknou, zpráva šabloně neodpovídá.

Popsaná pravidla je ještě třeba uzpůsobit s ohledem na různou povinnost segmentů, skupiny segmentů a příkazy. Shrňme proto pravidla pro přesuny teček následovně:

- Pokud se tečka po posunu nachází před povinným segmentem, zůstane tam.
- Pokud se nachází před nepovinným segmentem, zůstane a přidá další tečku za tento segment.
- Pokud se nachází na konci šablony, zpráva odpovídá šabloně.
- Pokud se nachází před začátkem povinné grupy segmentů, posune se do grupy.
- Pokud se nachází před začátkem nepovinné grupy segmentů, rozdvojí se – do grupy a za grupu

- Pokud se nachází před koncem grupy, rozdvojí se – na začátek grupy a za grupu.
- Pokud se nachází před příkazem _if, je tento příkaz zapsán do fronty příkazů dané tečky a tečka se posouvá za něj. O frontě příkazů viz dále.

Tyto kroky je potřeba opakovat tak dlouho, dokud nejsou všechny tečky před segmentem popisujícím segment zprávy, libovolně povinným či nepovinným. Jen pak jsme schopni zpracovat další segment, který přečteme ze zprávy.

Může se stát, že nalezený segment zprávy odpovídá hned dvěma či více segmentům v šabloně, před kterými je umístěna tečka. Jak tuto situaci vyřešit? Kde se vlastně v šabloně nacházíme? K této situaci by prakticky nemělo dojít. Struktura zprávy je jistě jednoznačná – to zaručují konkrétní normy jednotlivých zpráv, jinak by její význam nebylo možné interpretovat. Dostatek informací v šabloně je pak stejnou nutností – pisatel šablony na tento problém musí pamatovat při psaní šablony. Proto si můžeme dovolit při více možnostech zvolit první tečku, která se nachází před segmentem, kterému odpovídá aktuální segment zprávy. Segment zprávy pak se segmenty za ostatními tečkami ani porovnávat nemusíme.

Seznam všech teček v šabloně si proto celou dobu držme uspořádaný od těch nejblíže k začátku šablony po ty na konci. Budeme je tedy řadit podle čísla segmentu, před kterým se nacházejí.

Při opakovaných přesunech teček může snadno dojít k zacyklení. Pokud se například tečka nachází na začátku grupy, ve které se jinak nachází pouze jeden nepovinný segment, dojde při přesunu k přidání další tečky za tento segment. Jenže tím se nachází tečka před koncem grupy – je proto přesunuta za grupu a na začátek grupy. Tím se ale opět tečka nachází před oním nepovinným segmentem, dochází k zacyklení.

Proto je nutné udržet ve zvláštním seznamu všechna pole, na kterých už se tečka při přesunu nacházela, ale byla z tohoto místa odstraněna, protože další segment nepopisuje segment zprávy (jedná se o kraj grupy či příkaz). Na toto pole už tečka být umístěna nesmí. Pokud by se sem měla dostat, je tečka zrušena. Tento seznam je na konci každého úspěšného přesouvání teček smazán a vytvořen znovu.

Dále je nutno zamyslet se nad zpracováním příkazů. Nabízí se příkazy zpracovávat ihned, jakmile je tečka přes tyto příkazy přenesena. Jenže potom mohou být některé příkazy zpracovány navíc – co když tečka, která přes tento příkaz přešla, nakonec zanikne? Proto mohou být příkazy zpracovány pouze tehdy, kdy už byl nalezen další segment zprávy odpovídající určitému segmentu šablony. Proto při přesunech teček přidáváme příkazy do fronty příkazů dané tečky. Při zdvojení tečky dojde ke zkopírování celé fronty příkazů této tečky. Teprve tehdy, kdy byl segment za tečkou úspěšně porovnán s dalším segmentem zprávy, jsou provedeny všechny příkazy, které byly ve frontě této tečky. Tímto principem vlastně přidáváme příkazy „na dluh“, a teprve až si jsme jisti vlastní pozicí ve zprávě, doženeme všechny zanedbané příkazy.

Jakmile se dostane první tečka na konec šablony, můžeme s jistotou prohlásit, že zpráva odpovídala šabloně. Díky zpracování příkazů „na dluh“ a pravidlu výběru prvního segmentu při zpracování je možné vysledovat celý postup zpracování jen z pořadí spouštěných segmentů.

4.4.2 Porovnání segmentů zprávy se segmenty šablony

Každý segment šablony, který popisuje nějaký segment zprávy, musí mít schopnost porovnat se s tímto segmentem a určit, zdali mu tento segment odpovídá. Tento proces

probíhá trochu jinak u zpráv standardu EDIFACT a variabilního standardu inhouse, než u zpráv ebXML a konstantního inhouse.

4.4.2.1 Zprávy EDIFACT a variabilní inhouse

Variabilní segmenty standardu inhouse je možné považovat za segmenty standardu EDIFACT. Variabilní inhouse segmenty totiž mají uvedeno několik separátorů, které odpovídají významu jednotlivých separátorů EDIFACT. Proto pro jejich zpracování s výhodou použijeme stejný algoritmus.

Segmenty šablon standardu EDIFACT musíme zpracovávat znak po znaku. K tomuto zpracování se hodí zvláštní druh automatu, který čte segment šablony, zároveň ale zpracovává segment zprávy. Prioritu EDIFACT separátorů obsahuje Tabulka 2: Priorita separátorů EDIFACT, strana 15.

- Pokud se v šabloně vyskytl zprošťující znak ?, porovná jej se zprávou a přečte v šabloně další znak, který také porovná s znakem ve zprávě. Oba dva tyto znaky se musí vyskytovat také ve zprávě.
- Pokud se v šabloně vyskytl libovolný separátor, musí být libovolný separátor také ve zprávě.
- Pokud se v šabloně vyskytl silnější separátor než ve zprávě, přečte zprávu až do tohoto separátoru. Tyto znaky nevyhodnocuje.
- Pokud se v šabloně vyskytl slabší separátor než ve zprávě, přečte šablonu až do separátoru uvedeného ve zprávě. Pokud se na tomto úseku vyskytly libovolné znaky k porovnání se zprávou, segmenty si neodpovídají.
- Pokud se v šabloně vyskytl název proměnné v polích _PUBLIC či _PRIVATE, liší se zpracování pro standard EDIFACT a inhouse.
    - Standard EDIFACT přečte zprávu až do následujícího libovolného separátoru a tento úsek uloží do dané proměnné. Poté přečte zprávu až do stejného separátoru, jaký následuje v šabloně.
    - Standard inhouse rovnou přečte zprávu až do stejného separátoru, jaký byl uveden v šabloně. Do proměnné uloží celý tento úsek.
- Všechny ostatní znaky v šabloně porovnává se znaky v segmentu zprávy. Pokud jsou tyto znaky různé, segment zprávy neodpovídá segmentu šablony.

Pokud bychom při hledání libovolného separátoru narazili na konec zprávy či šablony, budeme tento konec považovat za nalezení separátoru. Úsek, který měl končit hledaným separátorem, tedy končí až s koncem segmentu zprávy, respektive šablony.

Segment zprávy odpovídá segmentu šablony ve dvou případech:

- Pokud porovnávání segmentů dospělo až na konec segmentu šablony
- Pokud segment zprávy sice skončil dříve než segment šablony, v šabloně už se ale dále nenacházely žádné další znaky, které by bylo třeba porovnat se zprávou.

4.4.2.2 Zprávy ebXML a konstantní inhouse

U zpráv standardů ebXML a variabilní inhouse je situace o něco jednodušší.

Pro porovnání segmentů zde stačí vyhodnotit pouze všechna porovnání atributů, která jsou přímo v segmentu šablony uvedena. V případe zpráv ebXML se jedná o porovnávání atributů či obsahu tagů, v případě zpráv inhouse se porovnávají znaky, jejichž umístění je zadáno přímo pořadím v segmentu. Pro strukturu segmentů šablony viz část 3.6: Popis segmentů šablony, strana 14.

4.5 Znakové kódování zpráv

Dalším problémem, který je třeba vyřešit, je kódování zpráv. Většina zpráv je přijata v 7-bitovém kódování podle normy ISO 646 (následník kódu ASCII). Některé zprávy jsou ovšem přijaty v 8-bitovém kódování EBCDIC, ačkoli tím porušují normu EDIFACT (ISO 9735). Tyto zprávy jsou naštěstí již na vstupní bráně zkonvertovány do standardního kódování ASCII.

Některé zprávy jsou ovšem přijaty v národních kódováních, zejména dle normy ISO 8859-2, znakové sady Windows-1250 či sady UTF-8. Proto je nutné zprávy číst ve stejném kódování v jakém byla zpráva napsána. Například pokusíme-li se číst zprávu zapsanou v kódování ISO 8859-2 jako zprávu UTF-8, nemusí se nám toto čtení vůbec provést. Některé kombinace znaků v ISO-8859-2 jsou v UTF-8 nepřípustné.

Samotná zpráva bohužel kódování (až na výjimky ve zprávách ebXML) neobsahuje. Proto je nutné informaci o kódování zprávy přečíst ze šablony či z konfiguračního souboru, jak bylo popsáno v odstavcích 4.3: Načtení šablony, strana 25, a 5.3: Formát konfiguračního souboru, strana 33. Nechť má kódování udané v šabloně vyšší prioritu, tedy kódování udané v konfiguračním souboru se použije až tehdy, nebylo-li kódování v šabloně zadáno.

Zprávy standardů EDIFACT a Inhouse tedy stačí přečíst právě v tom kódování, které jsme přečetli ze šablony, respektive z konfiguračního souboru.

Menší problém je nyní se zprávami standardu ebXML, které mohou obsahovat kódování také ve zprávě. Kódování uvedené ve zprávě můžeme přečíst pod libovolným kódováním založeným na 7-bitovém kódování ASCII. I kdyby zpráva byla ve víceznakovém kódování (například UTF-8), stále se nám ji podaří celou přečíst v libovolném 8-bitovém kódování. V případě kódování UTF-8 ji dokonce přečteme tak, že zde nebudou žádné neviditelné znaky (ASCII < 31) s výjimkou znaků <CR>, <LF>, mezery a tabulátoru. Blíže viz normu kódování UTF-8.

S ohledem na doporučení XML zde zvolme následující priority:

- Nebylo-li kódování uvedeno ani ve zprávě ani v šabloně, zvolme jako kódování UTF-8.
- Bylo-li kódování pouze v šabloně, použijme kódování ze šablony.
- Bylo-li kódování pouze ve zprávě, použijme kódování ze zprávy.
- Bylo-li kódování ve zprávě i v šabloně, použijme kódování ze zprávy.

Kódování uvedené v konfiguračním souboru se tedy při čtení zpráv standardu UTF-8 vůbec nevyužije. Místo něj je, v případě neznámého kódování, třeba použít kódování UTF-8, jak říká doporučení konsorcia W3C pro formát XML.

Celý proces pro načtení ebXML zpráv ve správném kódování se může zdát zbytečný. Doporučení pro formát XML totiž jasně specifikuje, že kódování musí být buď uvedeno v hlavičce XML dokumentu, nebo musí být celý dokument v kódování UTF-8. Bohužel zde se jedná o jeden z případů, kdy běžná praxe při používání ebXML zpráv nedodržuje XML doporučení. Byly přijaty zprávy, které, ačkoli v hlavičce uvedeno kódování neměly, byly uloženy v jiném kódování než v UTF-8. I tyto zprávy je nutné zpracovat.

Otázkou je také v jakém kódování zapíšeme výslednou nálepku. Jedna možnost je použít stejné kódování, jaké bylo použito ve zprávě. Druhá je pak vybrané informace ze zprávy zkonvertovat do jednotného kódování. Zkonvertujme proto všechny informace

použité v šabloně do nativního kódování platformy, na které právě instance Dispatchera běží. Toto je ostatně výchozí chování jazyka java při zápisu do souboru. Následné čtení informací z nálepky bude pak jednoduché.

4.6 Výroba nálepky

Posledním krokem zpracování každé zprávy je výroba nálepky. Jedná se pouze o uložení všech proměnných z pole _PUBLIC do souboru či databáze.

V případě uložení do souboru zvolme jednoduchý formát zápisu:

```
proměnná="hodnota"
```

Kde proměnná je název proměnné v poli _PUBLIC, hodnota je její hodnota uzavřená v uvozovkách. Všechny uvozovky v hodnotě proměnné nechť jsou zproštěny významu znakem zpětné lomítko, znaky zpětné lomítko nechť jsou zdvojeny.

4.7 Zpracování chyb

V průběhu práce Dispatchera může dojít k velkému množství různých chybových stavů. Zpracovávaná zpráva může být poškozena, v šabloně se může vyskytnout chyba, a v neposlední řadě může být chyba také v samotném kódu Dispatchera. Bohužel může dojít také ke mnoha nechtěným stavům, které nemohou být Dispatcherem rozpoznány jako chybové. Šablona může být příliš krátká, takže dojde ke zpracování některých zpráv podle této šablony, i když to původně nebylo zamýšleno.

Velice důležitým prvkem běhu Dispatchera je proto tvorba vyčerpávajících záznamů o zpracování. Tyto záznamy nazývejme **logy**. V každém logu se musí vyskytnout minimálně dostatek informací, aby bylo jasné ze kterých segmentů zprávy byla data v nálepce vykopírována. Také z logu musí vyplynout jasný postup, podle kterého postup práce probíhal. Díky těmto musí být možné celý proces zpracování zprávy zrekonstruovat.

Dojde-li k výskytu chybového stavu, je potřeba provést hned několik kroků. Do záznamu zpracování musí být zapsán přesný popis chyby i s umístěním, kde k této chybě došlo. Zpráva je poté zkopírována do speciálního adresáře, ve kterém čeká na kontrolu člověkem. Do tohoto adresáře je také vytvořena nálepka obsahující doposud získané informace. Chybový adresář může být periodicky kontrolován hlídacím modulem celého systému, aby byla na neočekávaný stav upozorněna obsluha systému.

Je nutné si uvědomit, že jakákoli nejistota při zpracování (například jedna chybná šablona) by měla vést k chybovému stavu. Extrakce špatných informací ze zprávy může být velice nebezpečná, a je proto lepší předat nejasnou zprávu obsluze systému, než se tuto zprávu snažit za každou cenu zpracovat. Obsluha systému poté může upravit šablony tak, aby příště dopadlo zpracování zprávy jednoznačně.

5 Uživatelská dokumentace

Podívejme se nyní na konkrétní způsob použití modulu Dispatcher.

Prvním krokem jeho používání je sepsání dostatečného množství šablon. Šablony musí být schopny pokrýt všechny zprávy, které přes konkrétní instanci modulu Dispatcher prochází. Pro znalost formátu šablon je nutné seznámit se minimálně s obsahem Kapitoly 3: Formát šablon, strana 11. Znalost způsobu zpracování ovšem také usnadní zápis šablon, před vlastním psaním šablon je proto dobré seznámit se s celou prací.

První šablony se ovšem dají napsat velice obecné. Například pro zprávu standardu EDIFACT stačí napsat jednu šablonu, která vyzvedne odesilatele a příjemce zprávy ze segmentu UNB. Jak již bylo řečeno, většinu zpráv se nám tímto způsobem identifikovat podaří. Bohužel na některé exempláře je tento způsob identifikace nedostatečný. V segmentu UNB například může být uveden pouze identifikátor subjektu zastupujícího reálného odesilatele. Abychom získali identifikace pravého tvůrce zprávy, musíme hlouběji do struktury zprávy. Konkrétní šablony pro tyto zvláštní případy ale stačí napsat až později.

5.1 Příprava spuštění programu

Předpokládejme tedy, že již máme napsány šablony pro zpracování většiny zpráv. Všechny tyto zprávy je nutné nakopírovat do jednoho adresáře.

Dále je nutné vytvořit soubor s pořadím zpracování šablon. V tom souboru je vždy název jedné šablony na jednom řádku, pořadí šablon je dáno pořadím názvů šablon v tomto souboru. Řádek s komentářem začíná znakem #, prázdné řádky se ignorují.

Poté je nutné upravit konfigurační soubor Dispatchera. Je nutné do něj zapsat konkrétní umístění adresáře pro příchozí zprávy, odchozí zprávy, záznamy zpracování (logy), adresář se šablonami a umístění souboru se seznamem šablon. Všechny tyto parametry jsou v příkladném konfiguračním souboru řádně zdokumentovány, dále jsou vypsány v části 5.3: Formát konfiguračního souboru, strana 33.

5.2 Spuštění programu

Nyní již lze spustit Dispatchera. Dispatcher vyžaduje právě jeden parametr, a to kompletní cestu ke konfiguračnímu souboru. Dispatcher zpracuje všechny zprávy, které se nacházejí ve vstupním adresáři v době jeho spuštění, a ukončí se.

Dispatcher je napsán v jazyce Java, proto pro své standardní spuštění vyžaduje Java Virtual Machine ve verzi alespoň 1.5. Spuštění pak musí probíhat právě přes tento virtuální stroj následujícím příkazem:

```
java -jar Dispatcher.jar /cesta/k/souboru/s/konfigurací
```

Při spuštění programu se do paměti zavede nejen kód Dispatchera, ale také celý kód virtuálního stroje, který je značně objemný. Modul Dispatcher je nutné spouštět velice často, aby přijaté zprávy dlouho neležely ve vstupním adresáři. Pro praktické použití se předpokládá dávkové spouštění Dispatchera jednou za určitou časovou jednotku. Aby se minimalizovala doba spuštění modulu, předpokládá se také kompilace Dispatchera do samostatně spustitelného souboru. K jeho spuštění pak Java Virtual Machine není potřeba.

5.3 Formát konfiguračního souboru

Na každém řádku konfiguračního souboru je uveden právě jeden parametr Dispatchera. Formát řádků konfiguračního souboru zvolme následovně:

- Bílé znaky na začátcích a koncích řádků nemají význam.
- Řádek začínající znakem # je považován za komentář.
- Prázdné řádky nemají význam.
- Na velikosti písmen parametrů nezáleží
- Zadání parametru má podobu parametr="hodnota", přičemž se povolují mezery kolem znaku =.

V konfiguračním souboru mohou být zadány následující parametry:

- InputDir – Kompletní cesta k adresáři, kde se nacházejí vstupní zprávy
- OutputDir – Kompletní cesta k adresáři, kam budou přesunuty úspěšně zpracované zprávy a vytvořeny nálepky
- LogDir Adresář, kde bude ke každé zprávě vytvořen soubor s logem
- TemplateDir – Adresář, kde se nacházejí jednotlivé šablony
- ErrorDir – Adresář, kam je přesunuta zpráva a vytvořena nálepky v případě výskytu chyby
- TemplateListFileName – Cesta k souboru obsahujícího seznam jmen všech souborů se šablonami
- DefaultEncoding – Výchozí kódování, které se použije při prvním čtení zpráv
- BufferSize – Velikost bufferu pro kopírování zpráv
- proměnné – Libovolné přiřazení hodnot do proměnných v poli _PUBLIC a _PRIVATE.

Pokud některé z těchto parametrů nejsou zadány, použijí se jako hodnoty prázdné řetězce, s výjimkou parametru BufferSize, kde se použije hodnota 4096.

6 Programátorská dokumentace

6.1 Systémové požadavky

Pro standardní běh vyžaduje Dispatcher na hostitelském stroji Java Runtime Edition (JRE) verze alespoň 1.5. Pro překlad také potřebujeme Apache Ant a Java Development Kit (JDK). Všechny nástroje jsou k dispozici pro většinu běžně využívaných platforem.

Java Runtime Edition a Java Development Edition jsou volně ke stažení z webových stránek firmy Sun http://java.sun.com. Apache Ant je možné získat ze stránek http://ant.apache.org.

6.2 Překlad a spuštění programu

Pro překlad využijeme již hotového skriptu build.xml a Apache Ant:

```
ant -f build.xml
```

Umístění, kde leží ant, musí být součástí systémové proměnné PATH. Jinak je nutno Ant spouštět s plnou cestou. Ant vygeneruje jednak zkompilované třídy v adresáři build, jednak hotový soubor Dispatcher.jar v adresáři dist.

Pro spuštění programu po překladu použijeme JRE:

```
java -jar dist/Dispatcher.jar /cesta/k/souboru/s/konfigurací
```

Pro smazání všech vytvořených adresářů lze použít následující příkaz. Vytvořené adresáře jsou build (se zkompilovanými třídami), dist (s výsledným souborem Dispatcher.jar) a doc (s vygenerovanou dokumentací):

```
ant -f build.xml clean
```

Pro vygenerování dokumentace struktury tříd do adresáře doc lze použít následující příkaz:

```
ant -f build.xml javadoc
```

6.3 Princip běhu

Princip běhu Dispatchera již byl nastíněn v kapitole 4: Princip práce Dispatchera, strana 24. Podívejme se nyní na konkrétní implementaci algoritmu v jazyce Java.

Jak známo, Java je objektově orientovaný jazyk, každý prvek programu je proto reprezentován určitou třídou.

6.3.1 Třídy pro reprezentaci zpráv a šablon

Podívejme se nyní na reprezentaci zpráv a šablon v programu.

Třída Message reprezentuje zprávu Každá z těchto tříd má potomky pro konkrétní standardy – XML, Inhouse a EDI. Potomci třídy Message znají strukturu zprávy a jsou schopni z ní tvořit segmenty. Aby mohla třída InhouseMessage vytvořit segment zprávy, potřebuje k tomu informace z příslušného segmentu šablony. Třídy XMLMessage a EDIMessage jsou schopny vytvořit segment bez dalších informací – konec segmentu je dán samotnou strukturou zprávy.

Třída Template reprezentuje šablonu. Potomci třídy Template jsou schopni načíst příslušnou hlavičku šablony a jednotlivé řádky šablony převést na seznam segmentů.

Potomek EDITemplate musí být schopen získat verzi EDI zprávy, aby věděl, jestli je znak hvězdička separátorem opakujících se segmentů. K tomu slouží následující metoda:

- `void setEDIVersion(int ver)`.

Zpráva i šablona se skládají ze segmentů. Pro segmenty zpráv EDIFACT a Inhouse nám stačí jediná třída MessageSegment, která reprezentuje celý segment jako pouhý řetězec. Pro segmenty zpráv XML potřebujeme složitější podtřídu XMLMessageSegment, která ukládá segment XML zprávy jako jeho název, množinu atributů a obsah.

Segmenty šablon je potřeba rozlišit. Každý segment šablony obsahuje minimálně následující dvě metody:

- `boolean correspondsWith(MessageSegment mesSeg)`
- `void executeSegment(MessageSegment mesSeg)`

První jmenovaná slouží k porovnávání segmentů, druhá pak ke spouštění segmentů podle zprávy. Vlastní porovnávání a spouštění se liší nejen pro jednotlivé standardy EDIFACT, ebXML a Inhouse, ale také pro konstantní a variabilní Inhouse zprávy. Proto existuje potomek třídy TemplateSegment pro každý druh zprávy.

Jelikož si je zpracování EDIFACT a variabilních inhouse segmentů velice podobné, existuje dále třída AbstractEDISegmentParser s potomky InhouseSegmentParser a EDISegmentParser. Tato třída obsahuje kód potřebný ke zpracování podobných částí EDI a variabilních Inhouse segmentů. Jednotlivé podtřídy pak obsahují kód potřebný pro zpracování částí, které je pro EDI a variabilní Inhouse zprávy rozdílné.

Mezi nejdůležitější funkce jednotlivých tříd patří následující:

Třída Message a podtřídy:

- `MessageSegment getNextSegment(TemplateSegment desc)` – metoda slouží k získání dalšího segmentu zprávy. InhouseMessage k tomu potřebuje segment šablony.

Třída Template a podtřídy:

- `TemplateSegment getSegmentAt(int pos)` – slouží k získání segmentu šablony na dané pozici

Třída TemplateSegment a podtřídy:

- `boolean correspondWith(MessageSegment mesSeg)` – slouží pro porovnání segmentu šablony se segmentem zprávy.
- `void executeSegment(MessageSegment mesSeg)` – Slouží pro spuštění segmentu, tedy vykopírování dat do proměnných.

6.3.2 Třída pro zpracování zpráv a šablon

Zprávy podle šablon porovnává třída Parser. Jejím úkolem je implementace algoritmu popsaného v odstavci 4.4: Porovnání a zpracování zprávy dle určité šablony (Parser), strana 27. Tento algoritmus je implementován v metodě:

- boolean parse(Message message, Template template)

Metoda vrací true, pokud pro zprávu byla nalezena šablona a zpráva byla zpracována, jinak vrací false.

Každá pozice v šabloně (tečka u segmentu) je reprezentována třídou ParserPoint. Tato třída obsahuje číslo segmentu, před kterým se tečka nachází. Také obsahuje seznam příkazů, které je nutné provést v případě, že je označený segment úspěšně porovnán se segmentem zprávy.

Jak bylo řečeno v odstavci 4.4: Porovnání a zpracování zprávy dle určité šablony (Parser), strana 27, všechny tečky je nutné udržet v uspořádaném pořadí. Třída ParserPoint je ve třídě Parser ukládána v množinách a seznamech, je proto nutné zde předefinovat několik základních metod, které definuje interface Comparable. Poté již budou tečky do interních struktur třídy Parser automaticky ukládány v požadovaném uspořádaném stavu. Jejich pořadí nezávisí na seznamu příkazů, které má tečka jako „dluh“.

- `public int hashCode()` – Vrátí HashCode čísla segmentu
- `int compareTo(Object o)` – Porovná tečku s jiným objektem
- `boolean equals( Object o )` – Zjistí, zdali se tečka rovná jinému objektu

6.3.3 Třída pro provádění příkazů šablony

O spouštění příkazů a segmentů šablony se stará třída Executor. Její hlavní metody jsou následující:

- `void executeCommand(TemplateSegment command)` – spustí příkaz a upraví podle něj obsah proměnných.
- `boolean returnCommandFound()` – vrací true, pokud byl nalezen příkaz _return v šabloně.
- `void executeTemplateSegment( TemplateSegment templateSegment, MessageSegment messageSegment)` – spustí segment šablony a zvýší jeho interní číslo pro nahrazování znaků # v segmentu.

6.3.4 Třída pro reprezentaci proměnných

Všechny proměnné v polích _PUBLIC a _PRIVATE jsou reprezentovány třídou Variables. Tato třída nabízí jednoduché rozhraní pro práci s příslušnými proměnnými:

- `setPrivateVariable(String key, String value)` – Přiřadí hodnotu proměnné z pole _PUBLIC
- `getPrivateVariable(String key)` – Získá hodnotu proměnné z pole _PUBLIC
- `setPublicVariable(String key, String value)` – Přiřadí hodnotu proměnné z pole _PRIVATE
- `getPrivateVariable(String key)` – Získá hodnotu proměnné z pole _PRIVATE

Kromě těchto funkcí nabízí třída Variables ještě dvě užitečné metody:

- saveAll()
- loadAll()

Metody slouží pro uložení a načtení obsahu všech proměnných. Tato funkcionalita přijde vhod při zpracovávání většího množství zpráv. Všechny proměnné na počátku zpracování zprávy můžeme uložit, zprávu zpracovat, proměnné vypsat do nálepky a pokračovat ve zpracování další zprávy.

Všechny proměnné jsou interně reprezentovány dvěma mapami:

- `HashMap<String, Pair<String, String> >`

Z hlediska jednoduchosti by bylo dobré, aby přiřazování hodnot jednotlivým proměnným nebralo v úvahu velikost písmen názvu proměnné. Ve výsledné nálepce bychom ale ocenili názvy proměnných různou velikostí písmen. Proto obě mapy používají jako hodnotu třídu Pair, která slouží kromě uložení vlastní hodnoty také pro uložení velikosti písmen názvu proměnné.

7 Závěr

Předložená práce popisuje návrh struktury šablon tak, aby byly schopny popsat většinu běžně zpracovávaných zpráv. Podařilo se nám také navrhnout algoritmus, kterým je možné zprávy podle těchto šablon porovnávat a extrahovat z nich hledané informace.

Bohužel nelze s jistotou tvrdit, že popsané řešení bude dostatečně funkční. Kdykoli se může objevit nový standard zpráv či nový zvyk v psaní zpráv standardu inhouse. Velkou část případných nových změn je možné dodatečně implementovat do stávajícího řešení. Neustálé přizpůsobování kódu aktuálním potřebám je nevyhnutelný proces.

Velkým úkolem je také zápis samotných šablon zpráv. Přepis všech používaných norem EDIFACT zpráv do potřebných šablon a přidání informací o umístění hledaných dat by bylo velice zdlouhavé. Naštěstí jsou k dispozici tyto normy v podobě, ze které je možné provést poměrně snadnou konverzi do šablon. Tuto konverzi je možné provést strojově – vlastní strukturu zpráv tak není nutné do šablon přepisovat.

Také struktura zpráv standardu ebXML je často dána dokumentem DTD či XML schéma. Jelikož tyto dokumenty mohou popisovat pouze strukturu XML zpráv, je nutné je zkonvertovat do šablon pro Dispatchera. Pro tento úkol je také možné vytvořit či využít automatizované nástroje.

Strukturu zpráv inhouse je ovšem nutné zapsat do šablon ručně. Ruční proces je zde nevyhnutelný – vlastní struktura často záleží na konkrétních zvyklostech a na konzultaci odesílací a přijímací strany. Naštěstí zprávy inhouse tvoří pouze menšinu zpracovávaných zpráv.

V budoucnu je potřeba celý systém nasadit do reálného provozu. Teprve velké množství zpracovaných zpráv přinese dostatek zkušeností se systémem. Na základě těchto zkušeností bude nutné upravit chování Dispatchera, aby byl co největším přínosem pro celý systém.

8 Obsah přiloženého CD

Kompletní zdrojový kód Dispatchera je k dispozici na přiloženém CD. Kromě samotného kódu se na disku nachází také potřebné nástroje, dokumentace a také tato práce v souboru PDF.

Přímo na disku se nacházejí následující adresáře:

documentation – Dokumentace k práci

examples – Příklady šablon, zpráv a konfiguračních souborů

nbproject – Pracovní projekt prostředí NetBeans

program – Zdrojové a přeložené souboru Dispatchera

tools – Různé potřebné nástroje

Následuje výpis obsahu jednotlivých adresářů

8.1 Adresář documentation

Adresář obsahuje tuto práci v následujících souborech:

- Dispatcher.pdf – formát Adobe PDF
- Dispatcher.odt – formát balíku Open Office

Dále obsahuje také vygenerovanou dokumentaci programem javadoc v adresáři javadoc. Vygenerovaná dokumentace obsahuje popis všech tříd a jejich metod. Obsahuje také popis všech parametrů těchto metod. Vygenerovaná dokumentace se hodí pro základní orientaci mezi třídami. Lze prohlížet po otevření souboru index.html v běžném webovém prohlížeči.

8.2 Adresář examples

Adresář examples obsahuje příklady všech souborů, se kterými modul pracuje. Tyto příklady jsou rozděleny do následujících podadresářů:

- interchanges – Příklady vstupních výměn
- stickers – Příklady vygenerovaných nálepek k přiloženým výměnám.
- templates – Příklady šablon, pomocí kterých byly zprávy zpracovány

Dále se v tomto adresáři nachází ještě dva soubory:

- configuration.txt – Příklad konfiguračního souboru modulu
- templateList.txt – Příklad souboru se seznamem šablon.

8.3 Adresář nbproject

Tento adresář obsahuje pracovní projekt z vývojového prostředí NetBeans. Používané verze byly 5.5, 6.0 a 6.1beta. Projekt je k práci přiložen pro snazší práci se zdrojovými soubory. Samotné prostředí se na CD nenachází, aktuální verzi lze stáhnout ze stránek http://www.netbeans.org.

Součástí tohoto projektu jsou také zdrojové soubory, vygenerovaná dokumentace a distribuční soubor Dispatcher.jar. Tyto soubory zde byly ponechány, jelikož jsou součástí pracovního projektu.

8.4 Adresář program

Adresář program obsahuje všechny potřebné soubory pro kompilaci a spuštění Dispatchera.

Jelikož se pro překlad programu používá nástroj Apache Ant, je zde umístěn soubor build.xml. Ten obsahuje postup překladu pro nástroj Ant.

Dále je zde několik podadresářů:

- build – obsahuje jednotlivé zkompilované třídy
- dist – obsahuje výsledný distribuční soubor Dispatcher.jar.
- doc – obsahuje vygenerovanou dokumentaci nástrojem javadoc.
- src – obsahuje soubory se zdrojovým kódem.

Pro překlad programu je nutný jen podadresář src a soubor build.xml. Všechny ostatní adresáře jsou vytvořeny nástrojem Ant.

8.5 Adresář tools

Adresář tools obsahuje několik nástrojů, které jsou nutné pro překlad a spuštění programu. Použití těchto nástrojů již bylo zmíněno v odstavci 6.2: Překlad a spuštění programu, strana 34. Pro spuštění postačí Java Runtime Edition (JRE), přo překlad je pak nutná Java Development Edition (JDK). Všechny přiložené nástroje jsou určeny pouze pro 32-bitový operační systém na bázi Unixu. Pro ostatní systémy mohou být tyto nástroje staženy ze stránek http://java.sun.com, respektive http://ant.apache.org.

V adresáři se nacházejí následující soubory:

- apache-ant-1.7.0-bin.zip – Zabalený nástroj Ant. Samotný nástroj se po rozbalení nachází v podadresáři bin
- jdk-6u6-linux-i586.bin – Skript, který nainstaluje JDK
- jre-6u6-linux-i586.bin – Skript, který nainstaluje JRE

9 Slovníček

Element segmentu – Jedna část segmentu, například úsek mezi znaky + v segmentu EDIFACT.

Identifikace zprávy – Proces, jehož výsledkem je dostatek informací, aby bylo zřejmé o jakou zprávu se jedná. Informace lze získat ze vstupního kanálu, nebo ze různých míst samotné zprávy.

Log – Záznam o zpracování zprávy Dispatcherem.

Nálepka – soubor informací sloužících k jednoduché identifikaci zprávy

Norma zprávy – Dokument popisující možnou strukturu zprávy, určuje typ zprávy

Segment šablony – Jeden úsek šablony reprezentující jeden segment zprávy.

Segment zprávy – Jeden úsek zprávy nesoucí informaci.

Segmenty si odpovídají – Segment šablony popisuje strukturu segmentu zprávy

Spuštění segmentu šablony – Spuštění procesu extrakce informací ze segmentu zprávy do interních proměnných v polích _PUBLIC a _PRIVATE

Standard zprávy – Obecný formát zprávy, například ebXML, EDIFACT, ...

Struktura zprávy – konkrétní pořadí segmentů ve zprávě, jejich význam

Šablona – Popis struktury zprávy

Typ zprávy – konkrétní podoba struktury zprávy, například faktura.

Vstupní kanál – Komunikační prostředek sloužící pro přijetí výměny do systému.

Výměna – Balík dat, který si vyměňují partneři.

Výstupní kanál – Komunikační prostředek sloužící pro odeslání výměny ze systému.

Zpráva – Identifikovaná a zpracovaná část výměny, data, která jsme schopni interpretovat. Seznam segmentů.

10 Literatura

1. ISO 9735: Electronic data interchange for administration, commerce and transport (EDIFACT) - Application level syntax rules
2. ISO 15000: Electronic business eXtensible Markup Language (ebXML)
3. Java tutorial (http://java.sun.com/docs/books/tutorial/)